Programowanie warstwy wizualnej gry

Krzysztof Gdawiec

UNIWERSYTET ŚLĄSKI
INSTYTUT INFORMATYKI

Zapytania

Zapytania to mechanizm pozwalający poprosić OpenGL o informację na temat tego, co się dzieje w potoku graficznym.

Zapytania w OpenGL reprezentowane są przez obiekty zapytań, więc najpierw musimy je wygenerować.

```
void glGenQueries(GLsizei n, GLuint* ids);
```

`n` to liczba identyfikatorów obiektów zapytań do wygenerowania, `ids` to tablica na wygenerowane identyfikatory.
Jeśli OpenGL nie będzie w stanie wygenerować identyfikatora, to zwróci wartość 0.

Do usuwania obiektów zapytań służy funkcja:

```
void glDeleteQueries(GLsizei n, const GLuint* ids);
```

Zapytania o okluzję

OpenGL nie śledzi w sposób automatyczny liczby rysowanych pikseli. Musi je jawnie policzyć i trzeba jawnie wskazać, kiedy ma zacząć liczyć.

```
glBeginQuery(GL_SAMPLES_PASSED, queryID);
```

Wartość `GL_SAMPLESS_PASSED` oznacza pytanie „ile próbek zdało test głębi?". Zmienna `queryID` to identyfikator obiektu zapytania.

Obiekt zapytania zliczający próbki, które mogą stać się widoczne (bo przeszły z powodzeniem test głębi) nazywamy zapytaniem o okluzję (ang. occlusion query).

Po rozpoczęciu zliczania próbek renderujemy (w standardowy sposób) obiekt, a OpenGL zlicza próbki.

Aby OpenGL dokonał podliczenia wszystkiego co zostało zrenderowane od rozpoczęcia zliczania wywołujemy:

```
glEndQuery(GL_SAMPLES_PASSED);
```

Po zakończeniu zapytania musimy pobrać jego rezultat:

```
glGetQueryObjectuiv(queryID, GL_QUERY_RESULT, &result);
```

`queryID` to identyfikator obiektu zapytania, `result` to zmienna, do której zapisany zostanie wynik zapytania.

Sprawdzając czy `result` ma wartość 0 możemy określić czy obiekt jest widoczny.

OpenGL działa jako potok i może posiadać zakolejkowanych wiele poleceń, które jeszcze nie zostały w pełni przetworzone.

Może się zdarzyć, że nie wszystkie polecenia rysowania wywołane przed ostatnim `glEndQuery` zakończyły swoje działanie.

W takiej sytuacji funkcja `glGetQueryObjectuiv` spowoduje, że wszystko co zostało przekazane do kolejki podczas zapytania zostanie zrenderowane. Zatem będziemy mieć opóźnienie.

Jeśli zapytanie o okluzję ma stanowić optymalizację wydajności, to takie coś jest niedopuszczalne.

W OpenGL mamy możliwość pobrania informacji czy przy próbie pobrania wyniku będziemy musieli jeszcze czekać na niedokończone zadania.

```
glGetQueryObjectuiv(queryID, GL_QUERY_RESULT_AVAILABLE, &result);
```

`queryID` to identyfikator obiektu zapytania, `result` to zmienna na wynik.
Jeśli będziemy musieli czekać na wynik zapytania o okluzję w `result` będzie wartość `GL_FALSE`. W przeciwnym przypadku `GL_TRUE`.

Mając taką informację możemy podjąć decyzję czy pobrać wynik zapytania o okluzję czy zająć się czymś innym.

Co możemy zrobić z informacją z zapytania o okluzję?

Najczęstszym zastosowaniem jest optymalizacja działania aplikacji przez unikanie wykonywania zadań.

Zamiast rysować złożony i kosztowny obiekt tworzymy jego uproszczoną wersję, np. prostopadłościan otaczający. Uruchamiamy zapytanie o okluzję, rysujemy uproszczony obiekt, kończymy zapytanie i pobieramy wynik. Jeśli w wyniku otrzymujemy, że obiekt nie spowodował narysowania ani jednego piksela na ekranie, to obiekt nie będzie widoczny i nie ma sensu rysować złożonego obiektu.

Przy renderowaniu uproszczonego modelu nie chcemy, aby pojawił się on w finalnej wersji sceny. W tym celu możemy użyć różnych podejść, np.

- używamy funkcji `glColorMask` do wyłączenia bufora koloru,
- używamy funkcji `glDrawBuffer` z ustawionym buforem na `GL_NONE`.

Po zastosowaniu każdej ze sztuczek należy ponownie przywrócić rysowanie po dokonaniu obliczeń.

```
glBeginQuery(GL_SAMPLES_PASSED, queryID);
renderSimplifiedObject(object);
glEndQuery(GL_SAMPLES_PASSED);
glGetQueryObjectuiv(queryID, GL_QUERY_RESULT, &result);
if( result != 0 )
  renderRealObject(object);
```

```
glBeginQuery(GL_SAMPLES_PASSED, queryID);
renderSimplifiedObject(object);
glEndQuery(GL_SAMPLES_PASSED);
glGetQueryObjectuiv(queryID, GL_QUERY_RESULT, &result);
if( result != 0 )
  renderRealObject(object);
```

```
GLuint result = 0;
glBeginQuery(GL_SAMPLES_PASSED, queryID);
renderSimplifiedObject(object);
glEndQuery(GL_SAMPLES_PASSED);

glGetQueryObjectuiv(queryID, GL_QUERY_RESULT_AVAILABLE,
    result);

if( result != 0 )
  glGetQueryObjectuiv(queryID, GL_QUERY_RESULT,&result);
else
  result = 1;

if( result != 0 )
  renderRealObject(object);
```

Rendering warunkowy

Rendering warunkowy umożliwia zebranie zestawu poleceń OpenGL w jedną paczkę i przesłanie jej do systemu wraz z obiektem zapytania oraz komunikatem „zignoruj to wszystko, jeśli wynik zapytania wynosi 0".

Aby rozpocząć sekwencję wywołań używamy:

```
glBeginConditionalRender(queryID, GL_QUERY_WAIT);
glBeginConditionalRender(queryID, GL_QUERY_NO_WAIT);
```

`queryID` to identyfikator obiektu zapytania, `GL_QUERY_WAIT` mówi, że OpenGL ma poczekać na wynik zapytania, `GL_QUERY_NO_WAIT` mówi, że OpenGL ma nie czekać na wynik zapytania tylko od razu renderować złożony obiekt.

Do zakończenia sekwencji używamy:

```
glEndConditionalRender();
```

```
    glBeginQuery(GL_SAMPLES_PASSED, queryID);
2   renderSimplifiedObject(object);
    glEndQuery(GL_SAMPLES_PASSED);
4
    glBeginConditionalRender(queryID, GL_QUERY_WAIT);
6   renderRealObject(object);
    glEndConditionalRender();
```

Transformacja sprzężenia zwrotnego

Wynik działania shadera wierzchołków, teselacji lub geometrii możemy zapisać w jednym lub kilku obiektach bufora, a następnie wykorzystać do odczytania lub jako dane wejściowe w następnych poleceniach rysowania. Mechanizm ten nosi nazwę transformacji sprzężenia zwrotnego.

Mechanizm ten jest nieprogramowalny, ale konfigurowalny.

Bufory służące do zapisania (nagrania) wyniku działania w/w shaderów nazywa się buforami sprzężenia zwrotnego.

Aby poinformować OpenGL o chęci nagrywania używamy:

```
void glTransformFeedbackVaryings(GLuint program, GLsizei count, const
GLchar* const* varying, GLenum bufferMode);
```

`program` – identyfikator programu cieniowania, `count` – liczba nagrywanych wyjść, `varying` – tablica z nazwami zmiennych wyjściowych do nagrywania, `bufferMode` – tryb w jakim ma nastąpić nagrywanie: `GL_SEPARATE_ATTRIBS` (każda zmienna trafia do osobnego bufora), `GL_INTERLEAVED_ATTRIBS` (zmienne trafiają do jednego bufora).

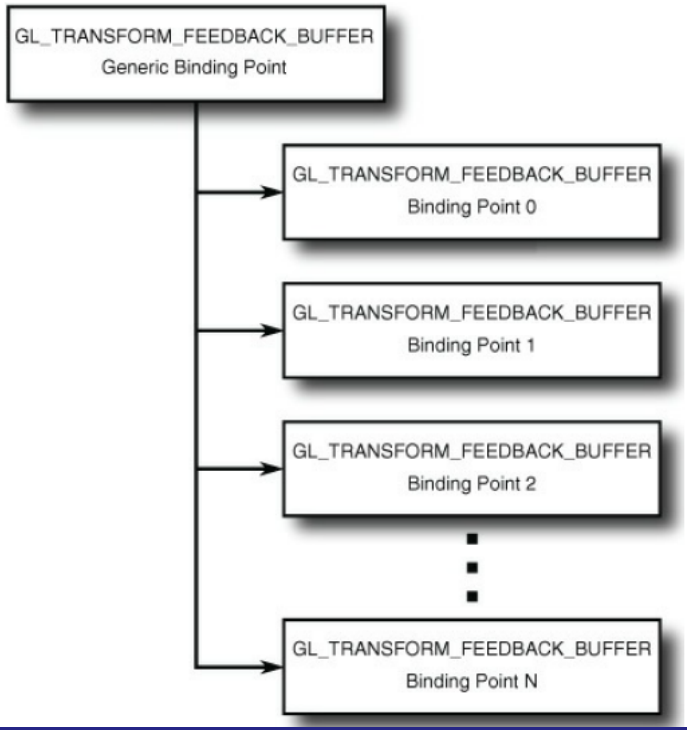
Nie wszystkie zmienne wyjściowe shadera trzeba umieszczać w buforze sprzężenia zwrotnego.

Użycie funkcji `glTransformFeedbackVaryings` musi wystąpić przed linkowaniem programu cieniowania za pomocą `glLinkProgram`.

Tworząc bufor musimy powiązać go z punktem dowiązania bufora sprzężenia zwrotnego.

```
    glGenBuffers(1, &buffer);
2   glBindBuffer(GL_TRANSFORM_FEEDBACK_BUFFER, buffer);
    glBufferData(GL_TRANSFORM_FEEDBACK_BUFFER, size, nullptr
        , GL_DYNAMIC_COPY);
```

Istnieje wiele punktów dowiązania.



Aby dołączyć bufor `buffer` do jednego z indeksowanych punktów dowiązań używamy funkcji:

```
glBindBufferBase(GL_TRANSFORM_FEEDBACK_BUFFER, index, buffer);
```

`index` – indeks punktu dowiązań.

Użycie `GL_SEPARATE_ATTRIBS` powoduje, że każde wyjście shadera znajdzie się w osobnym buforze. Zatem musimy określić wiele buforów jako bufory sprzężenia zwrotnego.

Liczba wyjść nagrywanych w osobnych buforach zależy od sprzętu i sterowników. Można ją pobrać za pomocą `glGetIntegerv` z parametrem `GL_MAX_TRANSFORM_FEEDBACK_SEPARATE_ATTRIBS`.

Możemy część wyjść zapisywać w buforze stosującym przeplatanie, a część w buforach zapisujących osobno poszczególne typy wartości.

Do tego celu służy „wirtualna” nazwa `gl_NextBuffer`, która informuje funkcję `glTransformFeedbackVaryings` o zamiarze przejścia do następnego bufora. `gl_NextBuffer` używamy w tablicy, w której podajemy nazwy zmiennych wyjściowych do nagrywania, np.

```
static const char* varying_names[] =
{
  "carrots",
  "peas",
  "gl_NextBuffer",
  "beans",
  "potatoes"
};
```

Uruchomienie sprzężenia zwrotnego odbywa się za pomocą funkcji:

```
void glBeginTransformFeedback(GLenum primitiveMode);
```

`primitiveMode` – informuje OpenGL jakiego typu geometrii się spodziewać: `GL_POINTS`, `GL_LINES`, `GL_TRIANGLES`.

W wywołaniu `glDrawArrays` lub innej funkcji rysującej typ geometrii musi odpowiadać temu podanemu w `primitiveMode`:

`primitiveMode`	Typ geometrii
`GL_POINTS`	`GL_POINTS`
`GL_LINES`	`GL_LINES`, `GL_LINE_STRIP`, `GL_LINE_LOOP`
`GL_TRIANGLES`	`GL_TRIANGLES`, `GL_TRIANGLE_STRIP`, `GL_TRIANGLE_FAN`

Możemy użyć również `GL_PATCHES` o ile shader na wyjściu zwraca geometrię odpowiedniego typu.

Tymczasowe wstrzymanie nagrywania:

```
void glPauseTransformFeedback();
```

Wznowienie nagrywania:

```
void glResumeTransformFeedback();
```

Wyłączenie trybu sprzężenia zwrotnego:

```
glEndTransformFeedback();
```

Wszystkie rysowania mające miejsce pomiędzy `glBeginTransformFeedback` a `glEndTransformFeedback` spowodują zapis danych w dowiązanych buforach sprzężenia zwrotnego.

W niektórych zastosowaniach sprzężenia zwrotnego chcemy jedynie wyliczyć pewne wartości wierzchołków. Ponieważ etap sprzężenia zwrotnego jest tuż przed rasteryzacją możemy wyłączyć rasteryzację i pozostałe etapy potoku:

```
glEnable(GL_RASTERIZER_DISCARD);
```

Aby ponownie włączyć rasteryzację używamy `glDisable` z tym samym parametrem.

Zapytania sprzężenia zwrotnego

Jeśli używamy wyłącznie shadera wierzchołków, to liczba wierzchołków umieszczona w sprzężeniu zwrotnym jest taka sama jak liczba wierzchołków wysłanych do OpenGL.

Jeśli używamy shadera geometrii, który może usuwać i dodawać wierzchołki, to liczba wierzchołków zapisywanych do bufora sprzężenia zwrotnego może być inna od liczby wierzchołków jakie do niego trafiły.

Aby poznać dokładną liczbę wierzchołków w buforze sprzężenia zwrotnego możemy skorzystać z zapytań.

Jako pierwszy parametr funkcji `glBeginQuery` możemy podać:

- `GL_PRIMITIVES_GENERATED` – zlicza wygenerowane prymitywy,
- `GL_TRANSFORM_FEEDBACK_PRIMITIVES_WRITTEN` – zlicza prymitywy zapisane w buforze sprzężenia zwrotnego.

Liczba wygenerowanych prymitywów i prymitywów zapisanych w buforze sprzężenia zwrotnego może być różna, np. gdy zabraknie miejsca w buforze, a prymitywy są nadal generowane.

Do pobrania wyniku zapytania, podobnie jak w przypadku zapytań o okluzję, używamy `glGetQueryObjectuiv`.

Obiekty transformacji sprzężenia zwrotnego

Generowanie identyfikatorów:

```
void glGenTransformFeedbacks(GLsizei n, GLuint* ids);
```

`n` – liczba identyfikatorów, `ids` – tablica na wygenerowane identyfikatory.

Dowiązanie obiektu:

```
void glBindTransformFeedback(GLenum target, GLuint id);
```

`target` – możliwa jest tylko wartość `GL_TRANSFORM_FEEDBACK`, `id` – identyfikator obiektu.

Usuwanie obiektów o podanych identyfikatorach:

```
void glDeleteTransformFeedback(GLsizei n, const GLuint* ids);
```

`n` – liczba usuwanych identyfikatorów, `ids` – tablica z identyfikatorami.

Sprawdzanie czy identyfikator `id` jest identyfikatorem obiektu transformacji sprzężenia zwrotnego:

```
GLboolean glIsTransformFeedback(GLuint id);
```

Obiekty transformacji sprzężenia zwrotnego mogą być użyte do automatycznego rysowania wierzchołków zapisanych wcześniej w buforze sprzężenia zwrotnego.

```
void glDrawTransformFeedback(GLenum mode, GLuint id);
```

`mode` – tryb rysowanych prymitywów, `id` – identyfikator obiektu transformacji sprzężenia zwrotnego. Przypomina wywołanie `glDrawArrays`, gdzie liczba wierzchołków do wyrysowania pobierana jest z obiektu.

```
void glDrawTransformFeedbackInstanced(GLenum mode, GLuint id, GLsizei
primcount);
```

`mode` – tryb rysowanych prymitywów, `id` – identyfikator obiektu transformacji sprzężenia zwrotnego, `primcount` – liczba kopii. Przypomina wywołanie `glDrawArraysInstanced`, gdzie liczba wierzchołków do wyrysowania pobierana jest z obiektu.